P1 1189182

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

July 02, 2004

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE.

APPLICATION NUMBER: *60/515,751*
FILING DATE: *October 30, 2003*
RELATED PCT APPLICATION NUMBER: *PCT/US04/09645*

By Authority of the
COMMISSIONER OF PATENTS AND TRADEMARKS

N. WOODSON
Certifying Officer

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH
RULE 17.1(a) OR (b)

PROVISIONAL APPLICATION COVER SHEET [37 CFR 1.53(c)]

This is a request for filing a PROVISIONAL APPLICATION under 35 U.S.C. §111(b) and 37 CFR 1.51(a)(2)

Date : October 30, 2003
Docket No. : 51457/JDC/R268
EXPRESS MAIL NO. EV 351346097US

Mail to: MAIL STOP PROVISIONAL PATENT APPLICATION

INVENTOR(S)/APPLICANT(S) (LAST NAME, FIRST NAME, MIDDLE INITIAL, RESIDENCE (CITY AND EITHER STATE OR FOREIGN COUNTRY))
(1) YAMADA, Kenshin; Los Angeles, CA (2) WANG, Ren; Los Angeles, CA (3) SANADIDI, M. Yahya; Los Angeles, CA (4) GERLA, Mario; Los Angeles, CA

_____ Additional inventors are being named on separately numbered sheets attached hereto.

TITLE OF THE INVENTION (280 characters max)
TCP WESTWOOD WITH AGILE PROBING

APPLICANT(S) STATUS UNDER 37 CFR § 1.27
_X_ Applicant(s) and any others associated with it/them under § 1.27(a) are a SMALL ENTITY

ENCLOSED APPLICATION PARTS
_17_ Specification (number of pages)
_____ Drawings (number of sheets)
_____ Assignment
_____ Other (specify):

FEE AND METHOD OF PAYMENT
_X_ A check for the filing fee of $ 80.00 is enclosed.
The Commissioner is hereby authorized to charge any fees under 37 CFR 1.16 and 1.17 which may be required by this filing to Deposit Account No. 03-1728. Please show our docket number with any charge or credit to our Deposit Account. **A copy of this letter is enclosed.**
_____ No filing fee enclosed.

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

_X_ No
_____ Yes, the name of the U.S. Government agency and the Government contract number are:

Please address all correspondence to **CHRISTIE, PARKER & HALE, LLP, P.O. Box 7068, Pasadena, CA 91109-7068, U.S.A.**

Respectfully submitted,
CHRISTIE, PARKER & HALE, LLP

By _____
John D. Carpenter
Reg. No. 34,133
626/795-9900

PROVISIONAL PATENT APPLICATION

TCP Westwood with Agile Probing:
Dealing with Dynamic, Large, Leaky Pipes

Kenshin Yamada, Ren Wang, M.Y. Sanadidi, and Mario Gerla
Computer Science Department, University of California, Los Angeles
Los Angeles, CA 90095, USA
{kenshin, renwang, medy, gerla}@cs.ucla.edu

Abstract

TCP Westwood (TCPW) has been shown to provide significant performance improvement over high-speed heterogeneous networks. The key idea of TCPW is use Eligible Rate Estimation (ERE) methods to set the congestion window* (cwnd) *and slow start threshold* (ssthresh) *after a packet loss. ERE is defined as the efficient transmission rate eligible for a sender to achieve high utilization and be friendly to other TCP variants. This paper presents TCP Westwood with Agile Probing (TCPW-A), a sender-side only enhancement of TCPW, that deals well with highly dynamic bandwidth, large propagation time/bandwidth, and random loss in the current and future heterogeneous Internet. TCPW-A achieves this goal by adding the following two mechanisms to TCPW: 1) When a connection initially begins or re-starts after a Timeout, instead of exponentially expanding* cwnd *to an arbitrary preset sthresh and then going into linear increase, TCPW-A uses Agile Probing, a mechanism that repeatedly resets* ssthresh *based on ERE and forces* cwnd *into an exponential climb each time. The result is fast convergence to a more appropriate* ssthresh *value. 2) In Congestion Avoidance, TCPW-A invokes Agile Probing upon detection of persistent extra bandwidth via a scheme we call Persistent Non-Congestion Detection (PNCD). While in Congestion Avoidance, Agile Probing is actually invoked under the following conditions: a) a large amount of bandwidth that suddenly becomes available due to change in network conditions; or b) random loss during slow start that causes the connection to prematurely exit the Slow-start phase.

Experimental results, both in ns-2 simulation and lab measurements using actual protocols implementation, show that TCPW-A can significantly improve link utilization over a wide range of bandwidth, propagation delay and dynamic network loading.

I. Introduction

TCP has been widely used in the Internet for numerous applications. The success of the congestion control mechanisms introduced in [15] and their succeeding enhancements has been remarkable. The current implementation of TCP Reno/NewReno runs in two phases: Slow Start and Congestion Avoidance. In Slow Start, upon receiving an acknowledgment, the sender increases the congestion window (*cwnd*) exponentially, doubling *cwnd* every Round-trip Time (RTT), until it reaches the Slow-start Threshold (*ssthresh*). Then, the connection switches to Congestion Avoidance, where *cwnd* grows more conservatively, by 1 packet every RTT (linearly). Then upon a packet loss, the sender reduces *cwnd* to half.

It is well known that the current TCP throughput deteriorates in high-speed heterogeneous networks, where many of the packet losses are due to noise and external interference over wireless links. Congestion control

schemes in current TCP assume that a packet loss is invariably due to congestion and reduce their congestion window by half, thus the performance deterioration.

When the Bandwidth-Delay Product (BDP) increases, another problem TCP faces is initial *ssthresh* setting. In many cases, initial *ssthresh* is set to an arbitrary value, ranging from 4k bytes to arbitrarily high (e.g., maximum possible value), depending on implementation under various operating systems. By setting the initial *ssthresh* to an arbitrary value, TCP performance may suffer from two potential problems: (1) if *ssthresh* is set too high relative to the network Bandwidth Delay Product (BDP), the exponential increase of *cwnd* generates too many packets too fast, causing multiple losses at the bottleneck router and coarse timeouts, with significant reduction of the connection throughput. (2) If the initial *ssthresh* is set too low, the connection exits Slow Start and switches to linear *cwnd* increase prematurely, resulting in poor startup utilization especially when BDP is large.

Dynamic bandwidth presents yet another challenge to TCP performance. In today's heterogeneous Internet, bandwidth available to a TCP connection varies often due to many reasons, including multiplexing, access control, and mobility [7]. First, the bandwidth available to a TCP flow is affected by other flows sharing the same bottleneck link. Second, in shared medium access networks, the bandwidth available to a TCP flow is highly variable depending on channel utilization and medium access protocol dynamics. Finally, hand-off and interference, among other factors in mobile networks, introduce significant bandwidth changes over time. Standard TCP versions can handle bandwidth decrease fairly well by its *cwnd* and *ssthresh* "multiplicative decrease" upon a congestion loss. However, if a large amount of bandwidth becomes available for reasons such as wide-bandwidth-consuming flows leaving the network, TCP may be slow in catching up (as will be shown below), particularly in Congestion Avoidance with its "additive-increase" mechanism, increasing *cwnd* only by 1 packet per RTT. As a result, link utilization can be lacking, particularly in large propagation times and highly dynamic large bandwidth, or what we might call "large dynamic pipes".

TCP Westwood (TCPW) has been proposed in [4, 21] and shown to provide significant performance improvement, better scalability and stability [24] over high-speed, heterogeneous networks. After a packet loss, instead of simply cutting *cwnd* by half as in standard TCP, TCPW-A resets *cwnd* along with *ssthresh* according to the TCPW sender's Eligible Rate Estimate (ERE), thus maintain a reasonable window size in case of random losses, and preventing over-reaction when transmission speed is high. TCPW relies on an adaptive estimation technique to determine a sender ERE at all times. The goal of TCPW is to estimate the connection eligible sending rate to achieve high utilization, without starving other connections. A brief overview of TCPW and ERE is given in Section II and detailed description can be found in [21].

In TCPW, ERE is only used to set *sshthresh* and *cwnd* after a packet loss. We realize that we can take advantage of ERE further when linear increase is too slow to ramp up *cwnd* (and simply setting a very high initial *ssthresh* is not a good idea as we will discuss later), as in cases of connection start-up and dynamic bandwidth aforementioned. In this paper, we propose TCP Westwood with Agile Probing (TCPW-A), a sender-side only enhancement of TCPW, that deals well with highly dynamic bandwidth, large propagation times and bandwidth, and random loss in the current and future heterogeneous Internet. TCPW-A achieves this goal by incorporating the following two mechanisms into basic TCPW algorithm:

The first mechanism is Agile Probing, which is invoked at connection start-up (including after a time-out), and after extra available bandwidth is detected. Agile Probing adaptively and repeatedly resets *ssthresh* based on ERE. Each time the *ssthresh* is reset to a value higher than the current one, *cwnd* climbs exponentially to the new value. This way, the sender is able to grow *cwnd* efficiently (but conservatively) to the maximum value allowed by current conditions without overflowing the bottleneck buffer with multiple losses – a problem that often affects traditional TCP. The result is fast convergence of *cwnd* to a more appropriate *ssthresh* value. In Slow Start, Agile Probing increases utilization of bandwidth by reaching "cruising speed" faster than existing protocols, this is especially important to short-lived connections.

The second mechanism concerns how to detect extra unused bandwidth. We realized that if a TCP sender identifies the newly materialized extra bandwidth and invokes Agile Probing properly, the connection can converge to the desired window faster than usual linear increase. This also applies in the case when a random error occurs during start-up, causing a connection to exit Slow Start prematurely and switch to Congestion Avoidance. In this paper, we propose a Persistent Non-Congestion Detection (PNCD) mechanism, which identifies the availability of persistent extra bandwidth in Congestion Avoidance, and invokes Agile Probing accordingly.

Experimental results, both in ns-2 simulation and lab measurements, show that TCPW-A can significantly improve link utilization under a wide range of system parameters.

The remainder of the paper is organized as follows. In Section II, we give an overview of TCP Westwood. In Section III and IV, We introduce the Agile Probing and Persistent Non-Congestion Detection (PNCD) mechanism respectively. Simulation Results to evaluate TCPW-A performance are provided in Section V. In Section VI, we describe the FreeBSD implementation of TCPW-A and report preliminary measurement results. Section VII discusses related work. Finally, section VIII discusses future work and concludes the paper.

II. TCP Westwood Overview

In TCP Westwood (TCPW), a sender continuously monitors ACKs from the receiver and computes its current Eligible Rate Estimate (ERE) [21]. ERE relies on an adaptive estimation technique applied to the ACK stream. The goal of ERE is to estimate the connection eligible sending rate with the goal of achieving high utilization, without starving other connections. Research on active network estimation [6] reveals that samples obtained by "packet pair" is more likely to reflect link capacity, while samples obtained by "packet train" give short-time throughput. In TCPW, the sender adaptively computes T_k, an interval over which the ERE sample is calculated. A ERE sample is computed by the amount of data in bytes that were successfully delivered in T_k. T_k depends on the congestion level, the latter measured by the difference between 'expected rate' and 'achieved rate' as in TCP Vegas. That is T_k depends on the network congestion level as follows:

$$T_k = RTT \times \frac{cwin/RTT_{min} - RE}{cwin/RTT_{min}}, \qquad (1)$$

where RTT_{min} is the minimum RTT value of all acknowledged packets in a connection, and RTT is the smoothed RTT measurement. The expected rate of the connection when there is no congestion is given by $cwnd/RTT_{min}$, while RE is the achieved rate computed based on the amount of data acknowledged during the

latest RTT, and exponentially averaged over time using a low-pass filter. When there is no congestion, and therefore no queuing time, *cwnd*/RTT_{min} is almost the same as RE, producing small T_k. In this case, ERE becomes close to a packet pair measurement. On the other hand, under congestion conditions, RE will be much smaller than *cwnd*/RTT_{min} due to longer queuing delays. As a result, T_k will be larger and ERE closer to a packet train measurement. After computing the ERE samples, a discrete version of a continuous first order low-pass filter using the Tustin approximation [22] is applied to obtain smoothed ERE.

In current TCPW implementation, upon packet loss (indicated by 3 DUPACKs or a timeout) the sender sets *cwnd* and *ssthresh* based on its current ERE. TCPW uses the following algorithm to set *cwnd* and *ssthresh*. For more details on TCPW and ERE, and its performance evaluation in high-speed, error-prone environments, please refer to [4] and [21].

```
if (3 DUPACKS are received)
  ssthresh = (ERE*RTTmin)/seg_size;
  if (cwnd >ssthresh) /*congestion avoid*/
    cwnd=ssthresh;
  endif
endif

if (coarse timeout expires)
  cwnd = 1;
  ssthresh =(ERE *RTTmin)/seg_size;
  if(ssthresh < 2)
    ssthresh = 2;
  endif
endif
```

III. AGILE PROBING

In this Section, we introduce Agile Probing scheme that improves TCP performance during start-up, and over large dynamic pipe with the help of PNCD. In Slow Start, Agile Probing is always used, while in Congestion Avoidance it is invoked only after PNCD detects persistent non-congestion. Below we discuss Agile Probing and present some preliminary results. In Section IV we present details of PNCD.

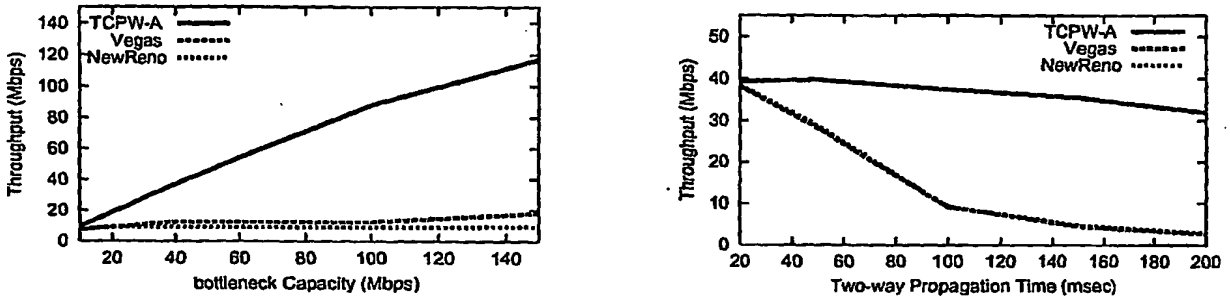
Agile Probing uses ERE to adaptively and repeatedly reset *ssthresh*. During Agile Probing, when the current *ssthresh* is lower than ERE, the sender resets *ssthresh* higher accordingly, and increases *cwnd* exponentially. Otherwise, *cwnd* increases linearly to avoid overflow. In this way, Agile Probing probes the available network bandwidth for this connection, and allows the connection to eventually exit Slow-start close to an ideal window corresponding to its share of path bandwidth. The pseudo code of the algorithm, executed upon ACK reception, is as follows:

```
if ( DUPACKS are received)
  switch to congestion avoidance phase;
else (ACK is received)
  if (ssthresh < (ERE*RTTmin)/seg_size)
     ssthresh = (ERE*RTTmin)/seg_size;/*reset ssthresh */
  endif
  if (cwnd >=ssthresh) /*linear increase phase*/
```

```
    increase cwnd by 1/cwnd;
  else if cwnd <ssthresh) /*exponentially increase phase*/
    increase cwnd by 1;
  endif
endif
```

By repeating cycles of linear increase and exponential increase, *cwnd* adaptively converges to the desired window in a timely manner, enhancing link utilization in Slow Start.

To assess the performance impact of Agile Probing we ran simulations with RTT based methods and compared the resulting throughput in the first 20 seconds of a connection lifetime to the corresponding throughput resulting from NewReno. All simulation results in this paper are obtained using the ns-2 simulator [19]. The initial *ssthresh* for NewReno is set to be Kbytes, a common default value among many implementations. The results are shown in Figure 1.



(a) Throughput vs. bottleneck capacity (b) Throughput vs. propogation time

Fig 1. Throughput comparison of TCPW-A, Vegas and NewReno (first 20 sec, rtt=100msec)

Figure 1(a) examines the throughput of TCPW-A, NewReno, and Vegas under bottleneck bandwidth varying from 10 to 150 Mbps (while fixing the round-trip time at 100ms). The results show that TCPW-A achieves much higher throughput, and scales well with bandwidth increase.

It is well known that RTT may considerably affect the startup performance. Figure 1(b) shows the throughput of TCPW-A, NewReno, and Vegas with RTT varying from 20ms to 200ms. The Bottleneck bandwidth is fixed here at 40 Mbps, and buffer size is set equal to the Bandwidth-Delay Product (BDP). The results show that TCPW-A scales well with RTT increase, while the performance of NewReno and Vegas deteriorates significantly with the increase of RTT.

Vegas' under-utilization over large pipes is mainly caused by its over-estimation of RTT, as we will state in more detail in Section VII. The poor startup performance of TCP NewReno, however, is due to its fixed initial *sshtresh* setting. When the pipe size (BDP) is small, i.e., the initial *ssthresh* is close to desired window (BDP), a NewReno connection can quickly fill up the pipe by exponentially increasing *cwnd* (Slow Start). On the other hand, when the pipe size becomes large, a NewReno connection prematurely exits Slow Start, slowing down the *cwnd* increase, resulting in low utilization.

In [13], Hoe proposes a method for setting the initial *ssthresh* to the product of delay and estimated bandwidth. The bandwidth estimation is calculated by applying the least-square estimation on three closely-spaced ACKs (similar to the concept of packet pair [26]). RTT is obtained by measuring the round trip time of the first segment transmitted. Hoe's modification enables the sender to get an estimation of the BDP at an early stage and set the *ssthresh* accordingly, thus avoiding switching to congestion avoidance prematurely. However, Hoe's method may be too aggressive when the buffer is not large enough, or network traffic is very dynamic.

We tested the startup behavior when another UDP connection joins a TCP connection in Slow Start phase. We ran simulations with one TCP connection starting at time 0 sec over a link with capacity of 40 Mbps, and a UDP flow with intensity of 20 Mbps starting at 0.5sec. Figure 2 shows that Hoe's method runs into multiple losses and finally timeouts. The reason is the setting of the initial *ssthresh* to 500 (BDP) at the very beginning of the connection, and the lack of adjustment to the change in network load later. In contrast, TCPW-A has a more appropriate (lower) slow-start exit *cwnd*, thanks to the continuous estimation mechanism which reacts to the new traffic and determines an eligible sending rate that is no longer the entire bottleneck link capacity.

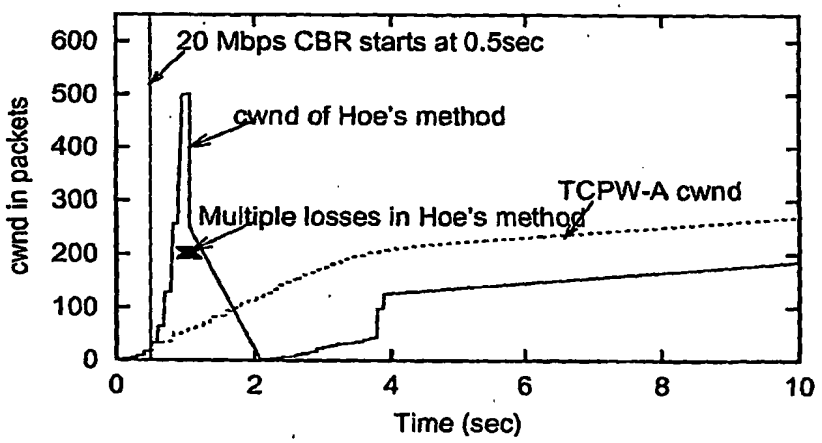


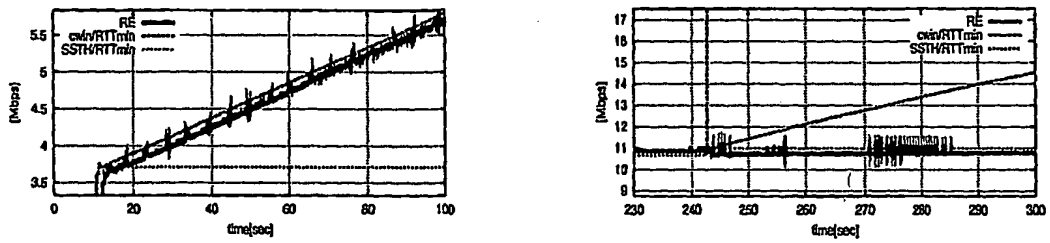
Fig 2. *cwnd* dynamic with UDP traffic joins in during start-up (bottleneck capacity = 40 Mbps, RTT=100ms, BDP =500 packets)

IV. Persistent Non-Congestion Detection

In this section, we present a Persistent Non-Congestion Detection (PNCD) mechanism that aims at detecting extra available bandwidth and invoking Agile Probing accordingly. In Congestion Avoidance, a connection monitors the congestion level constantly. If a TCP sender detects persistent non-congestion conditions, which indicates that the connection may be eligible for more bandwidth, the connection invokes Agile Probing to capture such bandwidth and improve utilization.

As described in section II, Rate Estimate (RE) is an estimate of the rate achieved by a connection. If the network is not congested and extra bandwidth is available, RE will increase as *cwnd* increases. On the other hand, if the network is congested, RE flattens despite of the *cwnd* increase. Figure 3(a) illustrates the

Expected Rate, which is equal to *cwnd*/RTT_{min}, and RE in non-congested path; while Figure 3(b) shows Expected Rate and RE under congestion. Also shown in these figures, are the *ssthresh/RTTmin* plots. Such values correspond to the "initial" expected rate. That is the expected rate when Congestion Avoidance was entered, or after a packet loss. From Figure 3(a), we see that RE follows *cwnd*/RTT_{min} continuously in non-congestion case. On the other hand, Figure 3(b) shows that RE does not grow and remains equal to *ssthresh*/RTT_{min} under congestion. In this case, RTT increases by an amount equal to the queuing time at the bottleneck. Then, this RTT's growth cancels out the *cwnd* increase, and keeps RE constant, as it should.



(a) Non congestion (b) Congestion

Fig 3. RE, *cwnd*, *ssthresh* dynamics

As mentioned before, *cwnd*/RTT_{min} indicates Expected Rate in no congestion and RE is the achieved rate. To be more precise, RE is the Achieved Rate corresponding to the Expected Rate 1.5 times RTT earlier[1]. Thus, we must use in a comparison, the corresponding Expected Rate, that is (*cwnd* -1.5)/RTT_{min}. RE tracks the Expected Rate in non-congestion conditions, but flattens, remaining close to the initial Expected Rate (*ssthresh* / RTT_{min}) under congestion. We define the Congestion Boundary as

$$CongestionBoundary = \beta \cdot ExpectedRate + (1-\beta) \cdot InitialExpectedRate \qquad 0 < \beta < 1 \qquad (2)$$

Figure 4 illustrates the relation among Congestion Boundary, Expected Rate and Initial Expected Rate with $\beta=0.5$.

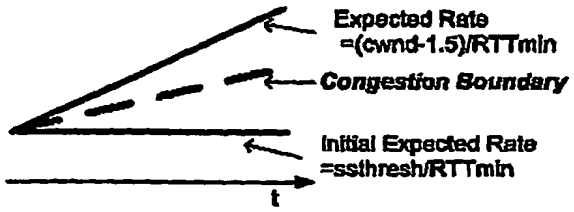


Fig 4. Congestion Boundary, Expected Rate and Initial Expected Rate with β=0.5

RE may fluctuate crossing above and below the Congestion Boundary. To detect persistent non-congestion, we use a (*non-congestion*) counter, which increases by one every time RE is above the Congestion Boundary and decreases by one if RE is below the Congestion Boundary. A pseudo code of the PNCD algorithm is as follows:

[1] The oldest acknowledged packet used for RE calculation is received one RTT before. Packets are traveled back and forth for RTT time period. Thus, the oldest ACK used for ERE calculation is sent two RTT before, and the newest ACK is sent one RTT before. On average, the ACK packets used for RE calculation are sent 1.5 RTT before. Then, the '*cwnd* at 1.5 RTT before' becomes (*cwnd* – 1.5).

```
if( in Congestion Avoidance except for the initial two RTT){
  if( RE >Congestion Boundary){
    no_congestion_counter++;
  else if(no congestion_counter > 0){
    no_congestion_counter--;
  if(no_congstion_counter > cwnd){
       re-start Agile Probing;
}else{
  no_congestion_counter = 0;
}
```

If the parameter β is greater than 0.5, the Congestion Boundary line gets closer to Expected Rate. We can make this algorithm more conservative by setting $\beta > 0.5$.

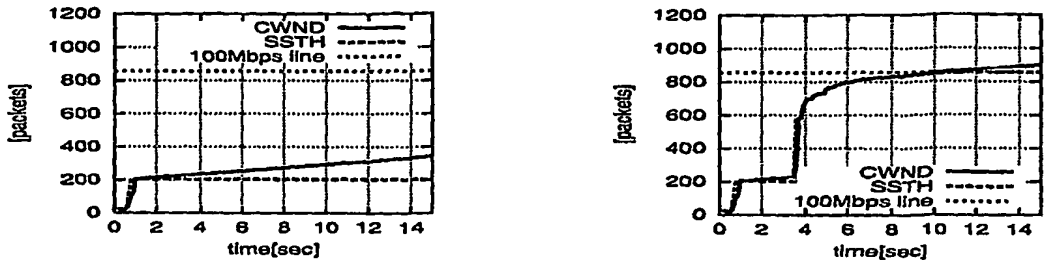
Even if the PNCD algorithm can accurately detect non-congestion, there is always the possibility that the network becomes congested immediately after the connection switches to Agile Probing phase. One such scenario is after a buffer overflow at the bottleneck router. Many of the TCP connections may decrease their *cwnd* after a buffer overflow, and congestion is relieved in a short time period. The PNCD in some connection may detect non-congestion and invoke Agile Probing. However, the erroneous detection is not a serious problem. Unlike exponential *cwnd* increase in Slow Start phase of NewReno, the TCP connection adaptively seeks the fair share estimate in Agile Probing mode. Thus, if the network has already been congested when a new Agile Probing begins, the "Agile Probing" connection will not increase *cwnd* much, and will go back to linear probing soon enough.

V. Simulation Results

In this section, we evaluate the performance of our TCP Westwood with Agile Probing (TCPW-A) algorithms in terms of throughput, friendliness, and window dynamics. The results show that TCPW-A exhibits significantly improved performance, yet remains friendly toward TCP NewReno, the de facto TCP standard over the Internet.

A. *Premature Exit from Slow Start*

Figure 5 shows *cwnd* dynamics under random packet loss during Slow Start. The bottleneck link bandwidth is 100Mbps, two-way propagation delay is 100msec, and the bottleneck buffer is equal to the BDP. When *cwnd/RTTmin* reaches 2Mbps, a packet is dropped (assumed to be random loss, which may happen in the early stage of a connection over Satellite or wireless links, as we observed in measurements). The connection exits Slow Start phase and enters Congestion Avoidance. Without PNCD, *cwnd* increases slowly, one packet every RTT, requiring more than 60 seconds for *cwnd* to reach BDP. With the help of PNCD, the TCPW-A connection detects persistent non-congestion within a few seconds, and then starts a new Agile Probing again. The throughputs of these 15 seconds simulations are 30.3Mbps without PNCD, and 88.8Mbps with PNCD and Agile Probing.



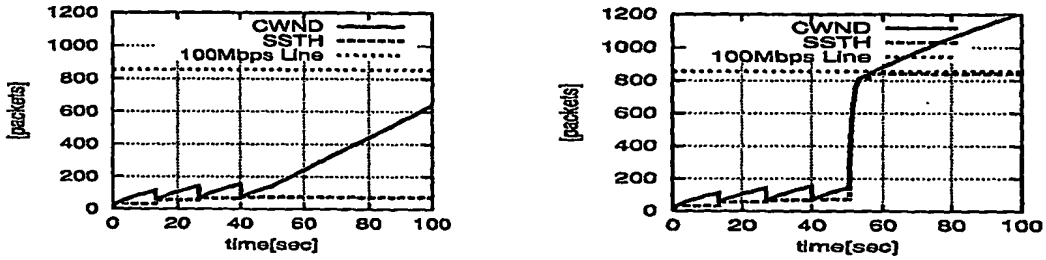
(a) TCP (without PNCD) (b) TCPW-A (with PNC and Agile Probing)

Fig 5. *cwnd* dynamic under a random packet loss at Slow Start phase

B. Dynamic bandwidth

To illustrate how TCPW-A behaves under dynamic bandwidth, Figure 6 shows *cwnd* dynamics when non-responsive UDP flows are gone from the path, causing extra bandwidth to become available.

The bottleneck link bandwidth is 100Mbps, two-way propagation delay is 100msec, and the bottleneck buffer is set to BDP.



(a) TCP (Without PNCD) (b) TCPW-A (with PNCD and Agile Probing)

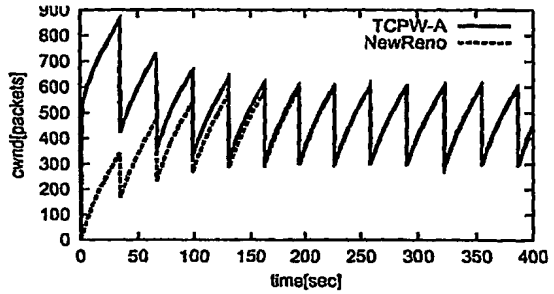
Fig 6. *cwnd* dynamic when dominant flows are gone from the bottlneck router

The non-responsive UDP flows have disappeared from the path around 50 seconds, and the remaining flow is eligible to use the newly materialized bandwidth. Without PNCD, the connection needs 60 seconds to reach BDP. On the other hand, PNCD detects the unused bandwidth within a few seconds, and a new Agile Probing phase makes instant use of this unused bandwidth possible! Note that dynamic bandwidth due to other reasons stated in Introduction will induce similar behavior that helps to improve utilization.

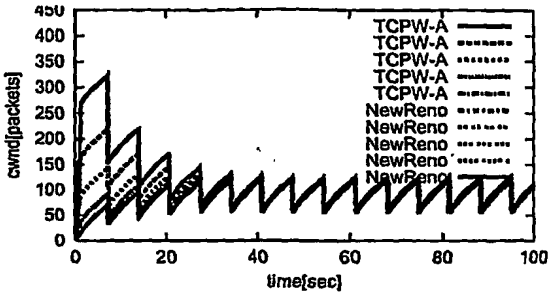
C. Friendliness

Since TCPW-A invokes Agile Probing, aggressively seeking unused bandwidth, the evaluation of TCPW-A friendliness is important. PNCD ensures that Agile Probing is invoked only in persistent non-congestion. Thus, if there are enough connections to fill the pipe, TCPW-A connections behave similar to TCPW. Thanks to good friendliness characteristics of TCPW, TCPW-A connections can effectively coexist with TCP NewReno connections over the same path. Figure 7 shows *cwnd* dynamics for TCPW-A and NewReno connections. The bottleneck link bandwidth is 100Mbps and a two-way propagation delay is 70msec.

In Figure 7 (a), one TCPW-A and one NewReno connection start running at the same time. The TCPW-A connection benefits initially by quickly reaching cruising speed, but they both reach the same *cwnd* after a few congestion losses. In Figure 7(b), five TCPW-A and five NewReno connections start running at the same time. The first started TCPW-A connection gets much more bandwidth initially, but all connections regardless of TCPW-A or NewReno reach fair share rate after a few packet losses.



(a) One TCPW-A and one NewReno connections



(b) 5 TCPW-A and 5 NewReno connections

Fig 7. *cwnd* dynamics between TCPW-A and NewReno (Bottlneck bandwidth = 100Mbps, RTT = 70msec)

D. Throughput Comparison Under Dynamic load

We evaluated the performance of TCPW-A under highly dynamic load conditions. In 20 minutes simulation time, we ran 100 connections, each with a lifetime of 30 seconds. The starting time of the connections are uniformly distributed over the simulation time. Thus, the set of connections is randomly spread out over 20 minutes simulation time, making the bandwidth available to a connection quite oscillating. The initial *ssthresh* of TCP NewReno is set to default value, 32k bytes.

Figure 8 shows Total throughput vs. bottleneck bandwidth. The total throughput is computed as the sum of throughputs of all connections. Two-way propagation delays are 70ms, and the bottleneck buffer size is set equal to the pipe size (BDP). When the bottleneck capacity is 10Mbps, TCP-A does not improve things, as NewReno can easily fill the small pipe. As the bottleneck link capacity increases, TCPW-A exhibits much better scalability than NewReno. At 150 Mbps, TCPW-A achieves at least twice as much throughput as NewReno does.

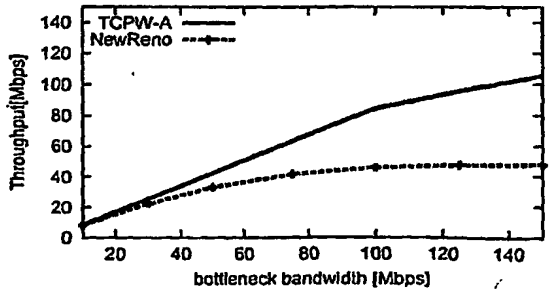


Fig 8. Link utilization vs bottleneck link capacity (RTT=70msec)

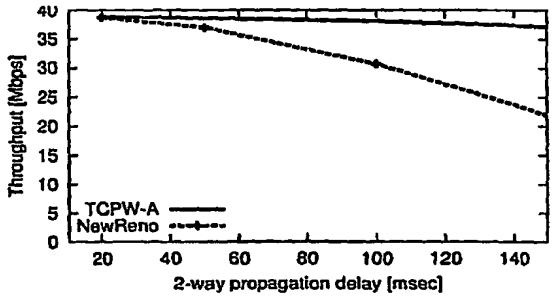


Fig 9. Link utilization vs 2-way propagation time (Bottleneck capacity = 45Mbps)

Figure 9 shows the total throughput vs. two-way propagation delay. The bottleneck bandwidth is 45Mbps, and the bottleneck buffer size is set equal to the pipe size (BDP). NewReno performance degrades as the propagation delays increase, showing lack of scalability to long propagation time. TCPW-A, on the other hand, scales well with increasing propagation times.

Note that there are two factors that account for TCP NewReno's inability to scale with bandwidth and RTT. One is its slow (linear) *cwnd* increase even when other connections leave the network and more bandwidth becomes available. The other factor is due to its small initial *ssthresh,* causing premature exit from Slow Start. One can argue that increasing the initial *ssthresh* easily solves this problem. However, a very large initial *ssthresh* may risk the connection into multiple packet losses and reduce the utilization even further. As also pointed out in [25], we think it is best to adaptively figure out the *ssthresh* value on the fly.

VI. Lab Measurements Results

To evaluate TCPW-A performance in actual systems, we have implemented TCPW-A algorithms, including Agile Probing and Persistent Non-Congestion Detection, on FreeBSD [11]. Lab measurements confirmed our simulation results, showing that TCPW-A behaves quite well in actual systems.

A. *Measurement setup*

In our lab measurement experiments, all PCs are running on FreeBSD Release 4.5 [11]. CPU clock tick is 10msec (default). We use Dummynet [8, 9] to emulate the bottleneck link. The bottleneck link (from PC router to TCP receiver) speed is set to 10Mbps. The propagation time is 400msec, and the router buffer is set equal to BDP (500Kbytes), equivalent to 62 packets. Queue management at the router is Drop-tail. We use Iperf [14] as a traffic generator. The receiver's advertised window is set large enough at 4Mbytes. The initial *ssthresh* for TCP NewReno is set to be 32Kbytes in our measurements.

B. *TCPW-A Convergence*

Figure 10 shows measured *cwnd* dynamics in TCPW-A and NewReno connections during Slow Start. NewReno enters Congestion Avoidance phase after *cwnd* reaches the initial *ssthresh* (32k), and *cwnd* increases linearly by one packet per RTT. *cwnd* reaches only 2Mbps, 20 % of the link capacity, for the first 25 seconds. On the other hand, in the TCPW-A connection, *cwnd* quickly converges to the link capacity by Agile Probing, We can also confirm that *cwnd* growth is not exponential throughout Agile Probing. Initially *cwnd* increases rapidly for the first 3 seconds, and then increases more slowly as the connection approaches the link capacity.

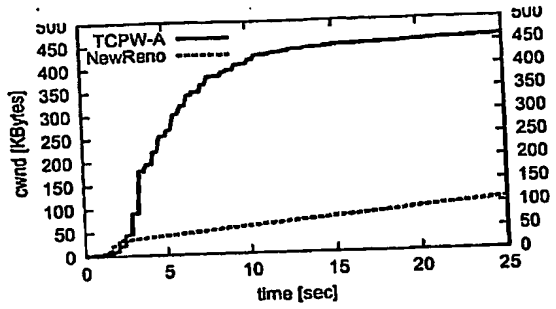


Fig 10. *cwnd* dynamics in TCPW-A and NewReno at Start-up (Lab Measurements Results)

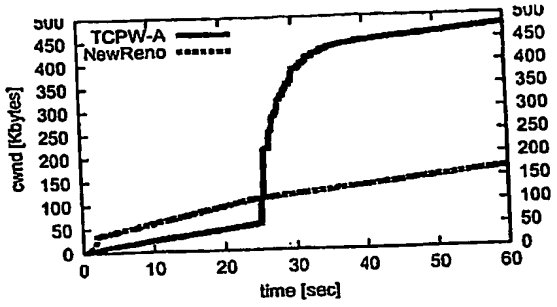


Fig 11. *cwnd* dynamics in TCPW-A and NewReno with prexisting flows in (0,25 sec)(Lab Measurements Results)

C. *PNCD Effectiveness*

Figure 11 compares *cwnd* dynamics in TCPW-A to that of NewReno as obtained from lab measurements. When a connection starts, the path is already filled with non-responsive UDP flows. NewReno enters Congestion Avoidance after *cwnd* reaches the initial *ssthresh* (32k). TCPW-A connection does not increase *cwnd* much since the bottleneck link has been already largely occupied. In fact, *cwnd* stays lower than that of NewReno. The dominant flows are finished after 25 seconds, and the entire link capacity becomes available to the remaining connection. The TCPW-A connection detects persistent non-congestion, and invokes Agile Probing to quickly capture the unused bandwidth. On the other hand, NewReno stays in Congestion Avoidance and linearly (and slowly) increases its *cwnd*.

VII. Related Work

A great deal of research effort has been made to enhance TCP performance for dynamic, large, leaky pipes. There are several other approaches to improving TCP scalability to large pipes that require only sender-side modification, including Scalable TCP [16], HighSpeed TCP [10], and Vegas-based FAST TCP [5]. In these schemes, as in traditional TCP, packet losses are exclusively treated as congestion signals. Compared to the previous schemes, TCPW-A is equipped with the ability to better handle random errors in high-speed heterogeneous networks. EXplicit Control Protocol (XCP) [17] is a well-designed congestion control scheme for high-speed, long delay networks. However, it requires cooperation from routers and receivers, making it difficult to deploy.

With the increase of short-lived web traffic [12], researchers realize that start-up performance is important, especially over large-pipes. The Agile Probing scheme in TCPW-A provides a realistic means to figure out the right *ssthresh* on the fly. A variety of other methods have been recently suggested in the literature to avoid multiple losses and to achieve higher utilization during Slow Start. A larger initial *cwnd*, roughly 4Kbytes, is proposed in [1]. This could greatly speed up transfers with only a few packets. However, the improvement is still inadequate when BDP is very large, and the file to transfer is bigger than just a few packets [23]. Fast start [20] uses *cwnd* and *ssthresh* cached from recent connections to reduce the transfer latency. The cached parameters may be too aggressive or too conservative when network conditions change. In [13], Hoe proposes

to set the initial *ssthresh* to the BDP estimated using packet pair measurements. This method can be too aggressive when the bottleneck buffer is not big enough, or many flows are co-existing. In [23], SPAND (Shared Passive Network Discovery) has been proposed to derive the optimal initial values for TCP parameters. SPAND needs leaky bucket pacing for outgoing packets, which can be costly and problematic in practice [2]. TCP Vegas [3] detects congestion by comparing the achieved throughput over a cycle of length equal to RTT, to the expected throughput implied by *cwnd* and *baseRTT* (the minimum RTT) at the beginning of a cycle. This method is applied in both Slow Start and Congestion Avoidance phases. During Slow Start, a Vegas sender doubles its *cwnd* only every other RTT, in contrast with Reno's doubling every RTT. A Vegas connection exits Slow Start when the difference between achieved and expected throughput exceeds a certain threshold. However, Vegas is not able to achieve high utilization in large bandwidth delay networks, due to its over-estimation of RTT [18].

To deal with dynamic bandwidth, TCP-EBN (Early Bandwidth Notification) is proposed in [7]. In TCP-EBN, a TCP sender increases or decreases its *cwnd* according to a bandwidth estimate that is sent to it from routers. Thus, this scheme relies on router cooperation and is therefore not an "end-to-end" approach to the problem at hand. Also, the router has to either keep per-flow state to get an accurate estimate, with the resultant scalability problem; or assumes that flows share bandwidth fairly, which is not true often. Comparing to this scheme, and XCP (which takes advantage of router feedback to swiftly adjust *cwnd* thus can handle dynamic bandwidth), TCPW-A only requires sender-side modification, thus much easier to deploy.

VIII. Conclusion and Future Work

In this paper we introduced TCP Westwood with Agile Probing (TCPW-A), a sender side only modification of TCP that addresses the issues of highly dynamic bandwidth, large delays, and random loss. Besides basic TCPW scheme presented in [4, 21], TCPW-A incorporates two new mechanisms: Agile Probing and Persistent Non-Congestion Detection. Agile Probing enhances probing during Slow Start and whenever non-congested conditions are detected. Agile Probing converges to more appropriate *ssthresh* values thereby making better utilization of large pipes, and reaching "cruising speeds" faster, without causing multiple packet losses. Another contribution of this work is the introduction of the Persistent Non-Congestion Detection (PNCD) method. PNCD is shown to be effective in detecting persistent non-congestion conditions, upon which TCPW-A invokes Agile Probing. The combination ensures that during Congestion Avoidance, TCPW-A can make quick use of bandwidth that materializes because of dynamic loads among other causes.

The results presented above were obtained using both simulation and laboratory measurements with actual implementation under FreeBSD operating system. The results show that TCPW-A works as well in actual systems as it does in simulation experiments.

In the future, we will evaluate TCPW-A further in terms of expanded friendliness, random loss, complex topologies and interaction with active queue management schemes. We also plan to move our measurements from the lab to the Internet and on satellite links.

REFERENCES

[1] M. Allman, S. Floyd and C. Patridge, "Increasing TCP's initial Window", INTERNET DRAFT, April 1998.

[2] A. Aggarwal, S. Savage, T.E. Anderson, "Understanding the Perfor-mance of TCP Pacing," In Proceedings IEEE INFOCOM 2000, Tel Aviv, Israel, March 2000.

[3] L.S. Brakmo and L.L. Perterson. TCP Vegas: End-to-End Congestion Avoidance on a Global Internet. IEEE Journal on Selected Areas in Communication, Vol. 13, Nov. 8, October 1995.

[4] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: bandwidth estimation for enhanced transport over wireless links," In Proceedings of Mobicom 2001, Rome, Italy, Jul. 2001.

[5] D.H. Choe, and S.H. Low, "Stabilized Vegas", In Proc. of IEEE/INFOCOM 2002, San Francisco, USA, April, 2002.

[6] C. Dovrolis, P.Ramanathan and D. Moore, "What Do Packet Dispersion Techniques Measure?," In Proceedings of Infocom 2001, Anchorage AK, April 2001.

[7] D. Dutta and Yongguang Zhang, "An Early Bandwidth Notification (EBN) Architecture for Dynamic Bandwidth Environments," Proceedings of IEEE International Conference on Communications (ICC'02), Apr 2002.

[8] Luigi Rizzo, "Dummynet: a simple approach to the evaluation of network protocols", ACM Computer Communication Review, 1997.

[9] IP Dummynet URL: http://info.iet.unipi.it/~luigi/ip_ dummynet/

[10] S. Floyd, "HighSpeed TCP for Large Congestion Windows", Internet draft draft-ietf-tsvwg-highspeed-01.txt, work in progress, August 2003.

[11] FreeBSD Project, URL: http://www.freebsd.org/

[12] L. Guo and I. Matta. The War between Mice and Elephants. In Proceedings of ICNP'2001: The 9th IEEE International Conference on Network Protocols, Riverside, CA, November 2001.

[13] J. C. Hoe, "Improving the Start-up Behavior of A Congestion Control Scheme for TCP", Proc. ACM SIGCOMM '96, pp. 270-280.

[14] Iperf Version 1.7.0, URL: http://dast.nlanr.net/Projects/Iperf/

[15] V. Jacobson, "Congestion avoidance and control," ACM Computer Communications Review, 18(4) : 314 - 329, Aug. 1988.

[16] T. Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks", Submitted for publication, December 2002.

[17] D. Katabi, M. Handley, and C. Rohrs, "Internet Congestion Control for Future High Bandwidth-Delay Product Environments." In Proceedings of Sigcomm 2002.

[18] S. Lee, B. G. Kim, and Y. Choi, "Improving the Fairness and the Response Time of TCP-Vegas," In Lecture Notes in Computer Science, Springer Verlag.

[19] NS-2 Network Simularor (ver.2.) LBL, URL: http://www.mash.cs.berkley.edu/ns/

[20] V.N. Padmamabhan and R.H. Katz, "TCP Fast Start: A Technique for Speeding Up Web Transfers", Proceedings of IEEE globecom'98, Sydney, Australia, Nov. 1998.

[21] R. Wang, M. Valla, M.Y. Sanadidi and M. Gerla, "Using Adaptive Bandwidth Estimation to provide enhanced and robust transport over heterogeneous networks", 10th IEEE International Conference on Network Protocols (ICNP 2002), Paris, France, Nov. 2002.

[22] K. J. Astrom, and B. Wittenmark, "Computer controlled systems," Prentice Hall, Englewood Cliffs, N. J., 1997.

[23] Y. Zhang, L. Qiu and S. Keshav, "Optimizing TCP Start-up Performance", Cornell CSD Technical Report, February, 1999.

[24] J. Chen, F. Paganini, R. Wang, M. Y. Sanadidi, M. Gerla " Fluid-flow Analysis of TCP Westwood with RED ", Proceedings of Globecom 2003, San Francisco, Ca., November 2003

[25] M. Allman, end2end-interest discussion group, July, 2003. http://www.postel.org/pipermail/end2end-interest/2003-July.txt

[26] S. Keshav A Control-Theoretic Approach to Flow Control. In Proceeding of ACM SIGCOMM' 1991, Pages 3-15, Sept. 1991.

Title: TCP Westwood with Agile Probing (TCPW-A): Handling Dynamic Large Leaky Pipes

Authors:

Ren Wang (contact author),	renwang@cs.ucla.edu,	UCLA,	Ph.D student
Kenshin Yamada,	kenshin@cs.ucla.edu,	UCLA,	Master student
Giovanni Pau,	gpau@cs.ucla.edu,	UCLA,	visiting scholar
M.Y. Sanadidi,	medy@cs.ucla.edu,	UCLA,	Faculty
Mario Gerla,	gerla@cs.ucla.edu,	UCLA,	Faculty

Abstract:

TCP has been widely used in the Internet for numerous applications.However, It is well known that the current TCP throughput deteriorates, due to its window reducing policy, in high-speed heterogeneous networks, where many of the packet losses are due to noise and external interference over wireless links.

When the Bandwidth-Delay Product (BDP) increases, another problem TCP faces is initial ssthresh setting. By setting the initial ssthresh to an arbitrary value, TCP performance may suffer from two potential problems: (1) if ssthresh is set too high, the exponential increase causes multiple losses at the bottleneck router and timeouts, with significant reduction of the connection throughput. (2) If the initial ssthresh is set too low, the connection exits Slow Start prematurely, resulting in poor startup utilization.

Dynamic bandwidth presents yet another challenge to TCP performance. In today's heterogeneous Internet, bandwidth available to a TCP connection varies often due to many reasons, including multiplexing, access control, and mobility.

TCP Westwood (TCPW) has been proposed and shown to provide significant performance improvement, better scalability and stability over high-speed, heterogeneous networks. After a packet loss, instead of simply cutting cwnd by half as in standard TCP, TCPW-A resets cwnd along with ssthresh according to the TCPW sender's Eligible Rate Estimate (ERE), thus maintain a reasonable window size in case of random losses, and preventing over-reaction when transmission speed is high. TCPW relies on an adaptive estimation technique to determine a sender ERE at all times. The goal of TCPW is to estimate the connection eligible sending rate to achieve high utilization, without starving other connections.

In TCPW, ERE is only used to set sshthresh and cwnd after a packet loss. We realize that we can take advantage of ERE further when linear increase is too slow to ramp up cwnd, as in cases of connection start-up and dynamic bandwidth aforementioned. We propose TCP Westwood with Agile Probing (TCPW-A), a sender-side only enhancement of TCPW, that deals well with highly dynamic bandwidth, large propagation times and bandwidth, and random loss in the current and

future heterogeneous Internet. TCPW-A achieves this goal by incorporating the following two mechanisms into basic TCPW algorithm:

The first mechanism concerns how to detect extra unused bandwidth. We realized that if a TCP sender identifies extra bandwidth (either during the connection startup or due to the dynamic bandwith), and invokes Agile Probing properly, the connection can converge to the desired window faster than usual linear increase. We propose a Persistent Non-Congestion Detection (PNCD) mechanism, which identifies the availability of persistent extra bandwidth, and invokes Agile Probing accordingly.

The second mechanism is Agile Probing, which is invoked after extra available bandwidth is detected. Agile Probing adaptively and repeatedly resets ssthresh based on ERE. Each time the ssthresh is reset to a value higher than the current one, cwnd climbs exponentially to the new value. This way, the sender is able to grow cwnd efficiently (but conservatively) to the maximum value allowed by current conditions without overflowing the bottleneck buffer with multiple losses - a problem that often affects traditional TCP. This is especially important to short-lived connections.

Experimental results, both in ns-2 simulation and lab measurements, show that TCPW-A can significantly improve link utilization under a wide range of system parameters.

TCP Westwood with Agile Probing(TCPW-A)

Handling Dynamic Large Leaky Pipes

Ren Wang, Kenshin Yamada, Giovanni Pau, M.Y. Sanadidi, and **Mario Gerla**
{renwang,kenshin,gpau,medy,gerla}@cs.ucla.edu

Problem Definition

- Leaky Pipes: packet loss due to error
  - Unjustified *cwnd* cut
  - Premature Slow Start exit
- Large Pipes: Large capacity and long delay
  - Control scheme may not scale
- Dynamic Pipes: Dynamic load/changing bandwidth
  - Linear increase limits efficiency

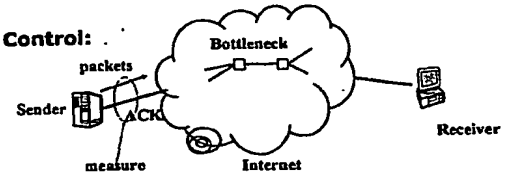
Proposed Solution

Sender-side only enhancement:

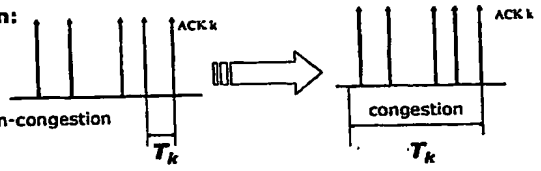
- TCP Westwood
- Persistent Non-Congestion Detection:
  - Detect extra unused bandwidth
  - Invoke Agile Probing
- Agile Probing: Probe efficiently

TCP Westwood (TCPW) and Eligible Rate Estimate (ERE)

TCPW Control:



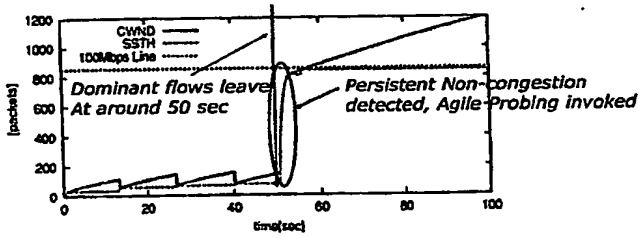
ERE Adaptation:



- Network viewed as blackbox;Estimation done on sender
- After dup-acks:
  - *cwnd* and *ssthresh* ← ERE * RTTmin
- After a timeout:
  - *ssthresh* ← ERE * RTTmin, *cwnd* ← 1

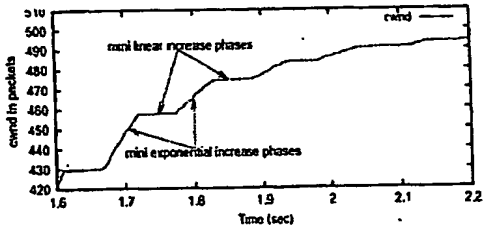
- ERE sample:Calculated by bytes delivered in interval T_k
  - Congestion level decided by expected rate and achieved rate
  - Light Congestion: short Tk,(packet-pair like)
  - Heavy Congestion: long Tk, (packet-train like)
- Using discrete low pass filter to get smoothed ERE

Persistent Non-Congestion Detection(PNCD)



- Objective: Detect unused bandwidth/invoke Agile Probing
- Observe Achieved Rate (AR) and Expected Rate (ER)
- If AR follows ER for a considerably long time -> ***PNC,*** indicating extra unused bandwidth->Agile Probing invoked

Agile Probing



- Objective: Guided by ERE, converge *faster* to *more appropriate* ssthresh
  - adaptively and repeatedly resets *ssthresh* to ERE*RTTmin
  - Exponentially increase cwnd if ssthresh >cwnd
  - Linearly increase cwnd if ERE < ssthresh
  - Exit Agile Probing when packet loss is detected

P rformance Evaluation